Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Lab 9 and Homework 9:
    - reopened till this Friday @ 11:59 pm
  - Homework 10: this Friday 12/2 @ 11:59 pm
  - Lab 11: Monday 12/5 @ 11:59 pm
  - Homework 11: Friday 12/9 @ 11:59 pm
  - Assignment 3: ~~Monday 11/28~~ Friday 12/9 @ 11:59 pm
  - Assignment 4: Friday 12/9 @ 11:59 pm

Previous Lecture

- Dynamic Programming Examples
  - Subset Sum
  - Edit Distance
  - Longest Common Subsequence

This Lecture

- Back to Graph Algorithms
- Network Flow Problem

Muddiest Points

- **Q: I would like another example of the edit distance algorithm. I did not quite understand what was going on too well**
- Sure!

Muddiest Points

- **Q: Subset Sum Problem**
- Let’s have another example!

Muddiest Points

- **Q: LCS**
- Let’s have another example!

Problem of the Day: Finding Bottlenecks

- Let's assume that we want to send a large file from point A to point B over a computer network as fast as possible over multiple network links if needed
- Input:
  - A computer network
    - Network nodes and links
    - Links are labeled by link capacity in Mbps
  - Starting node and destination node
- Output:
  - The maximum network speed possible for sending a file from source to destination

Defining network flow

- Consider a directed, weighted graph G(V, E)
  - Weights are applied to edges to state their *capacity*
    - c(u, w) is the capacity of edge (u, w)
    - if there is no edge from u to w, c(u, w) = 0
- Consider two vertices, a *source s* and a *sink t*
  - Let’s determine the maximum flow that can run from s to t in the graph G

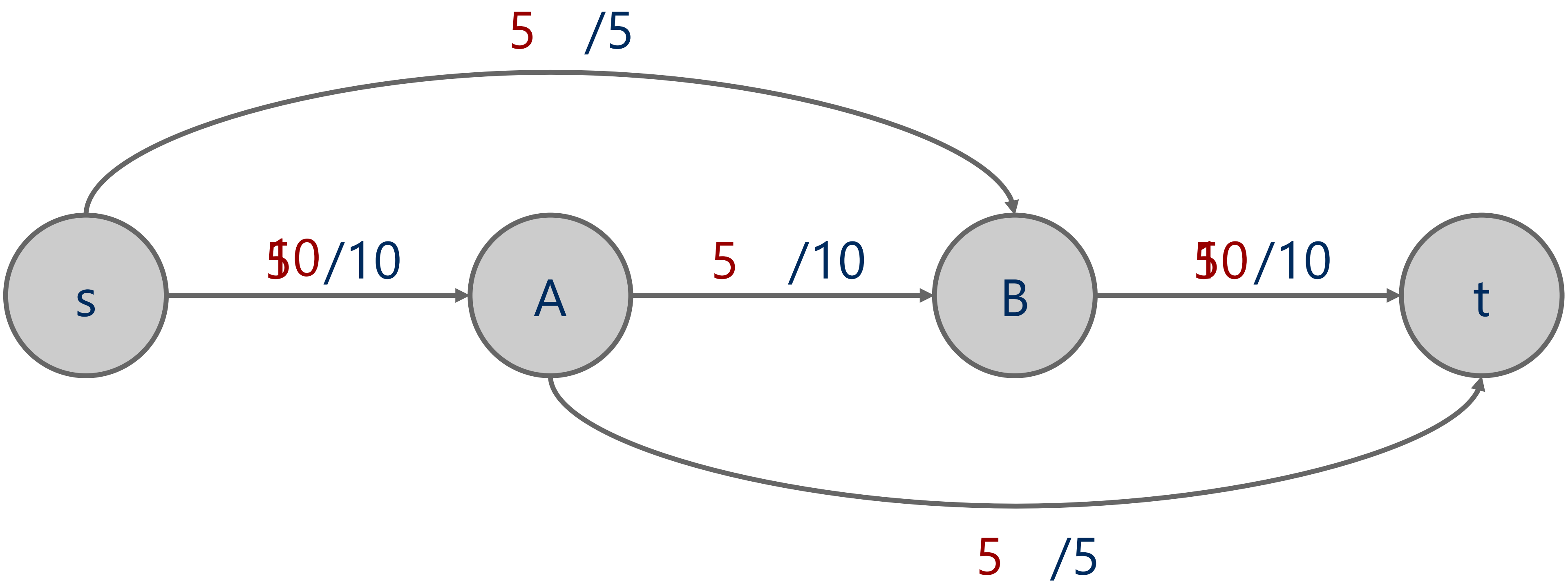
Flow

- Let the f(u, w) be the amount of flow being carried along the edge (u, w)
- Some rules on the flow running through an edge:
  - $\forall (u, w) \in E \; f(u, w) <= c(u, w)$
  - $\forall u \in (V - \{s,t\}) \; (\Sigma_{w \in V} f(w, u) - \Sigma_{w \in V} f(u, w)) = 0$

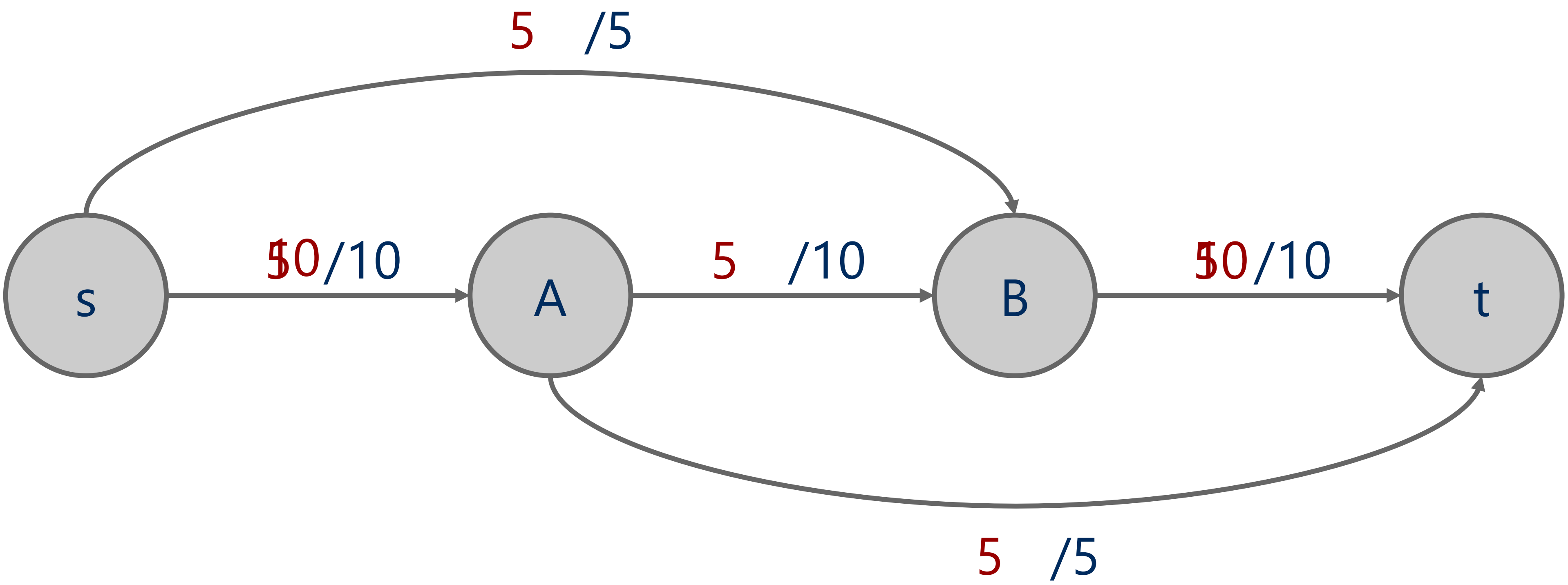
Ford Fulkerson

- Let all edges in G have an allocated flow of 0
- While there is path *p* from s to t in G s.t. all edges in p have some *residual capacity* (i.e., $\forall(u, w) \in p \; f(u, w) < c(u, w)$):
  - (Such a path is called an *augmenting path*)
  - Compute the residual capacity of each edge in p
    - Residual capacity of edge (u, w) is c(u, w) - f(u, w)
  - Find the edge with the minimum residual capacity in p
    - We'll call this residual capacity *new_flow*
  - Increment the flow on all edges in p by new_flow

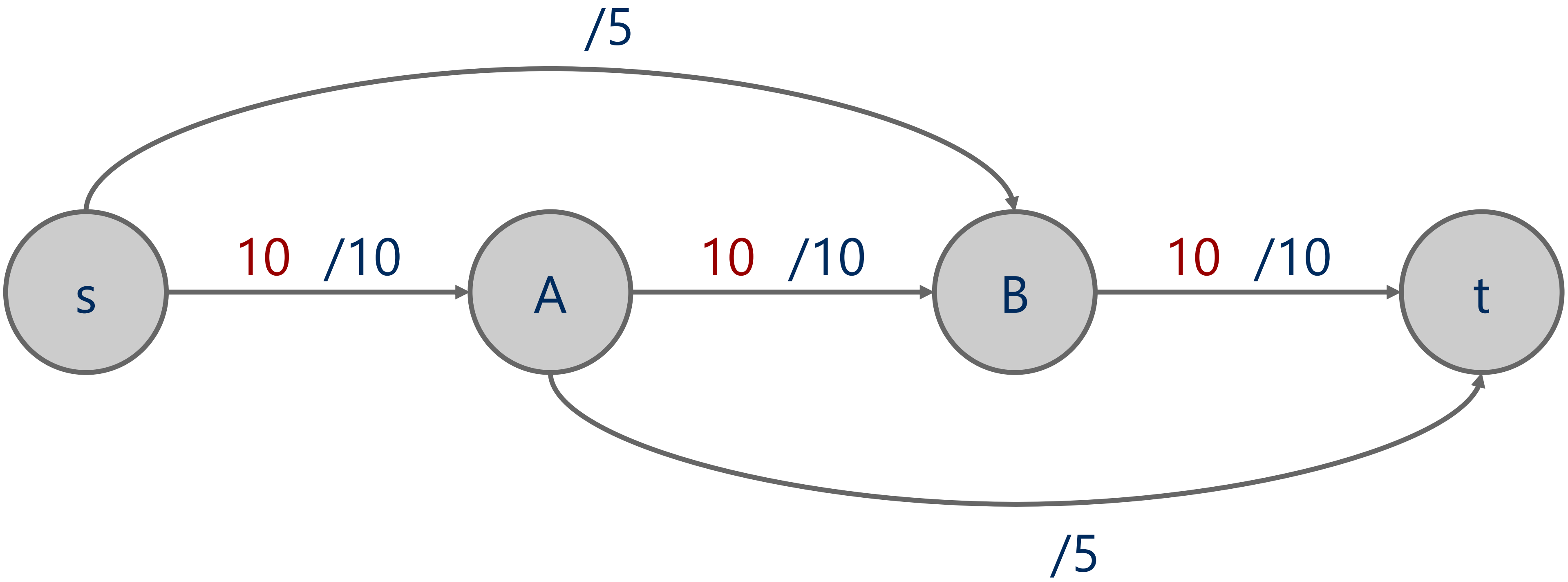
Ford Fulkerson example



Ford Fulkerson example



Another Ford Fulkerson example



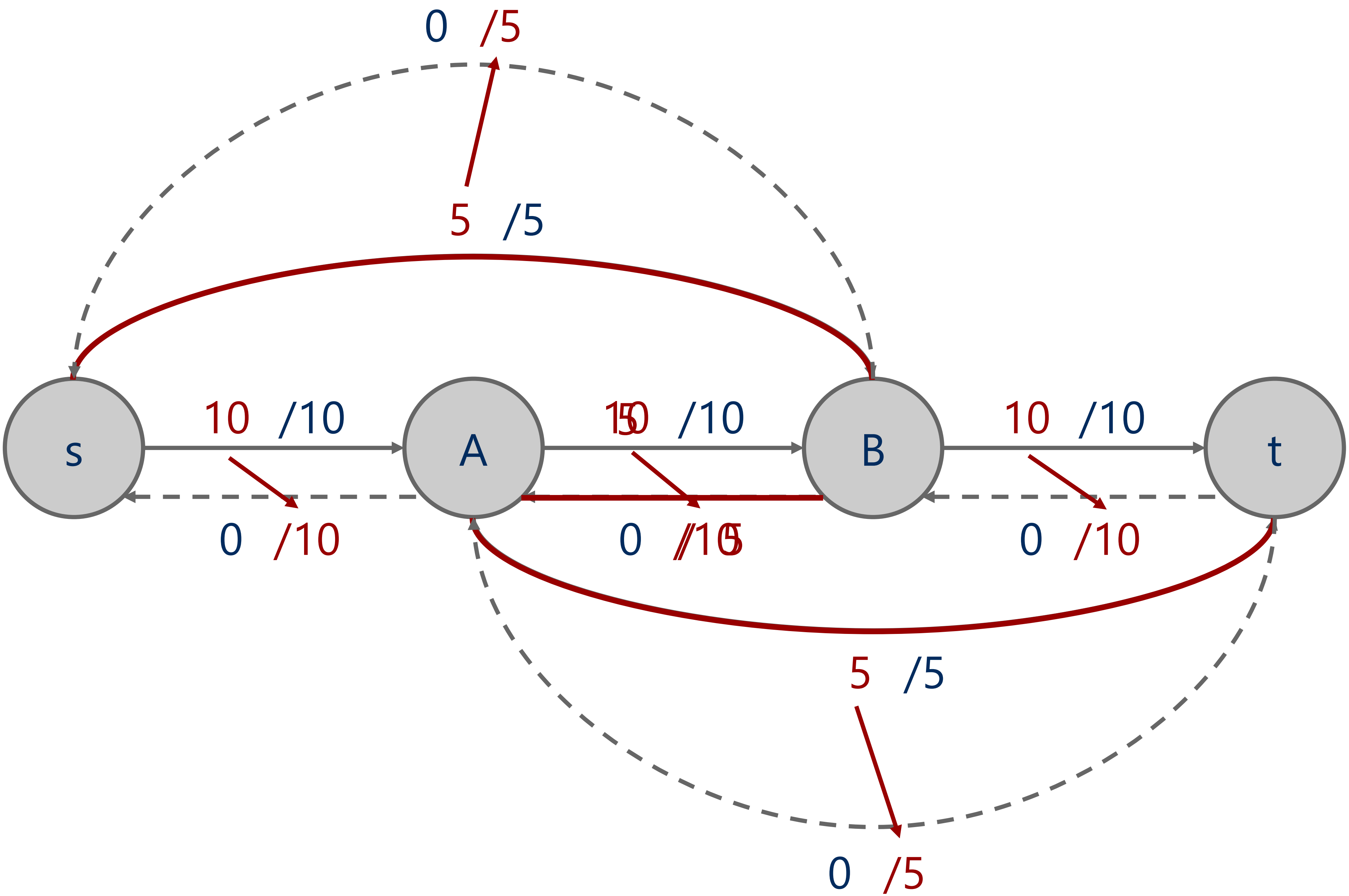
Expanding on residual capacity

- To find the max flow we will have to consider re-routing flow we had previously allocated
  - This means, when finding an augmenting path, we will need to look not only at the edges of G, but also at *backwards edges* that allow such re-routing
    - For each edge $(u, w) \in E$, a backwards edge (w, u) must be considered during pathfinding if $f(u, w) > 0$
      - The capacity of a backwards edge (w, u) is equal to $f(u, w)$

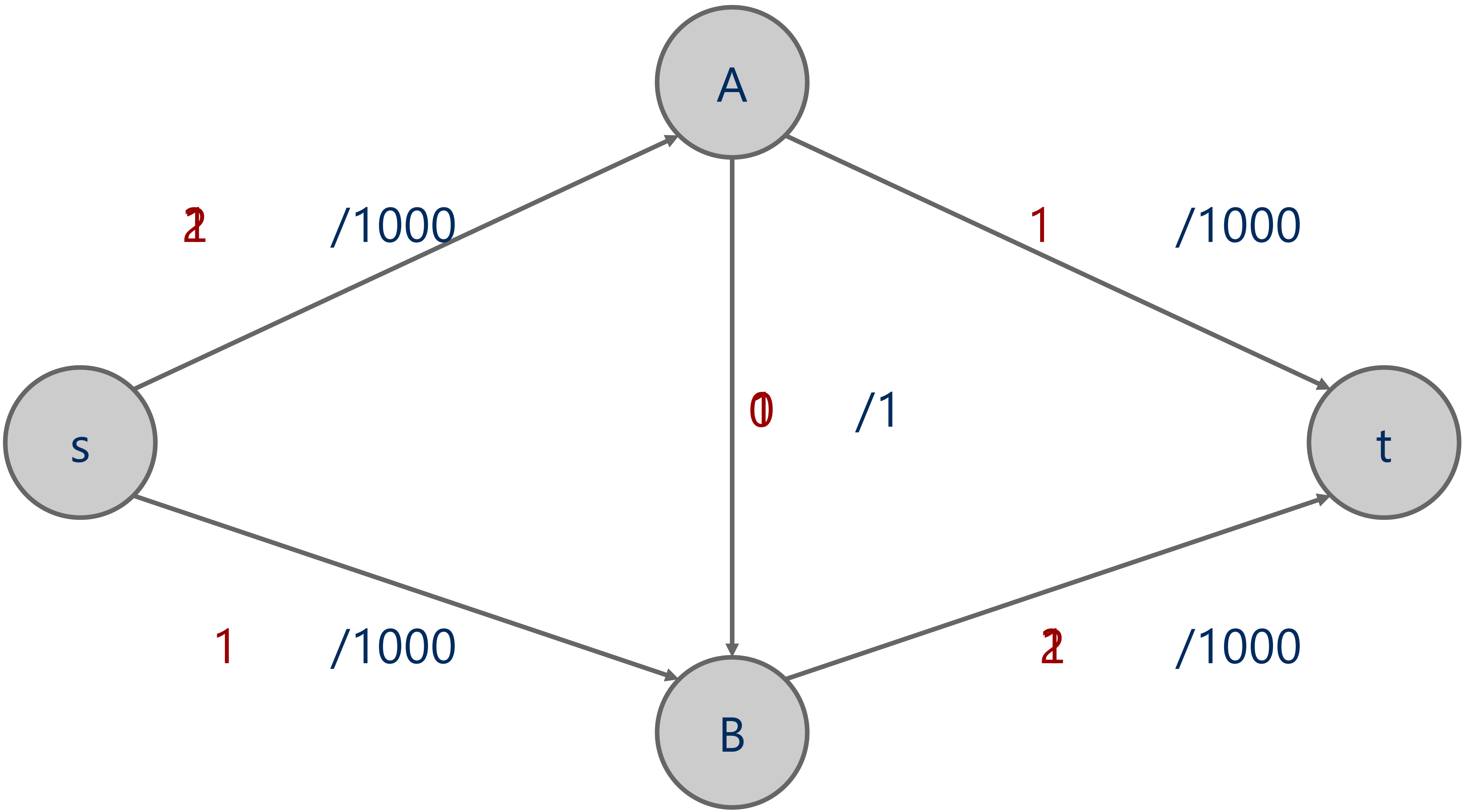
The residual graph

- We will perform searches for an augmenting path not on G, but on a residual graph built using the current state of flow allocation on G
- The residual graph is made up of:
  - V
  - An edge for each $(u, w) \in E$ where $f(u, w) < c(u, w)$
    - (u, w)'s mirror in the residual graph will have 0 flow and a capacity of $c(u, w) - f(u, w)$
  - A backwards edge for each $(u, w) \in E$ where $f(u, w) > 0$
    - (u, w)'s backwards edge has a capacity of $f(u, w)$
    - All backwards edges have 0 flow

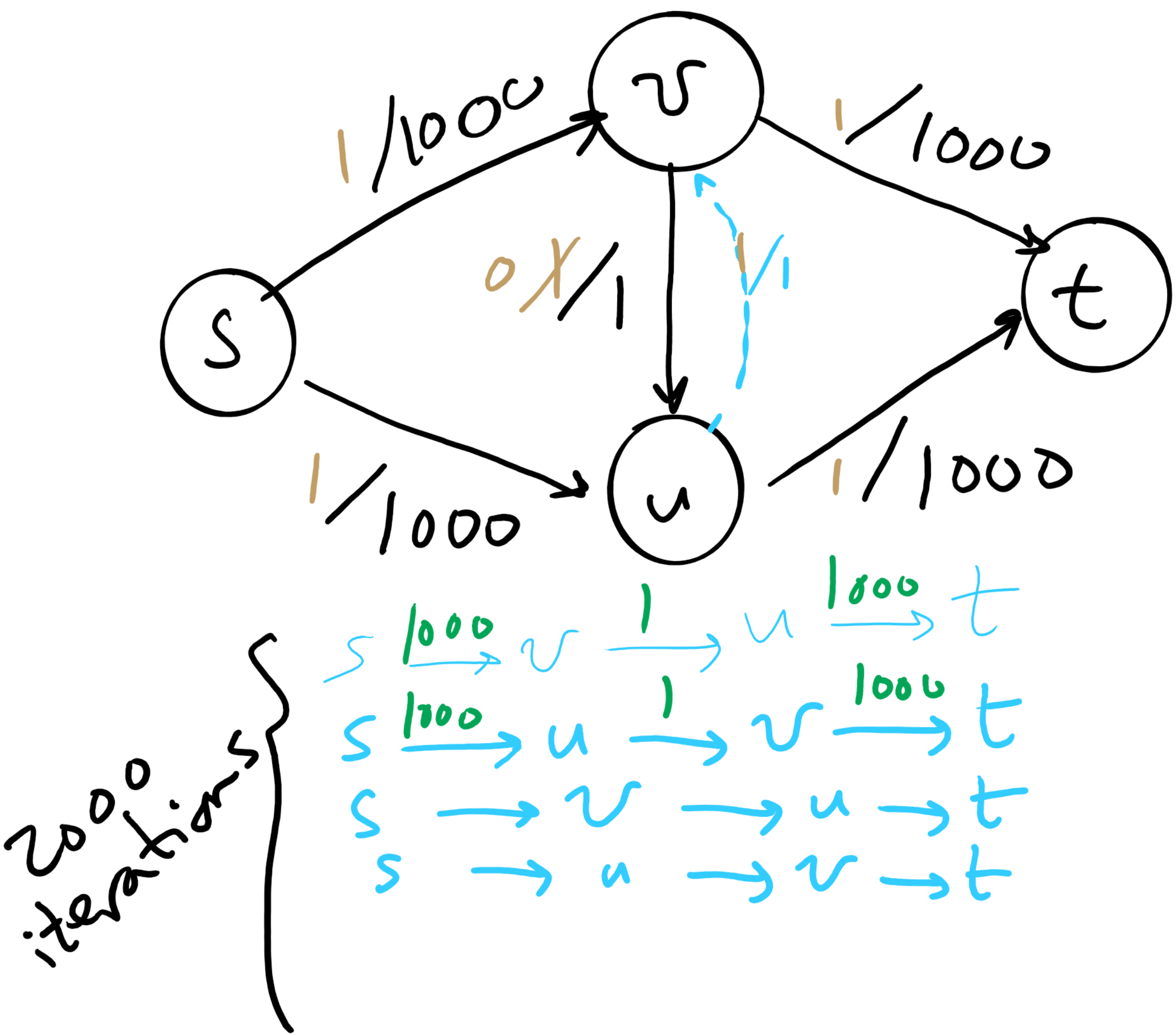
Residual graph example



Another example



Worst-case runtime of Ford-Fulkerson



Worst-case Runtime of Ford-Fulkerson

$$\Theta(|f| * (e+v))$$

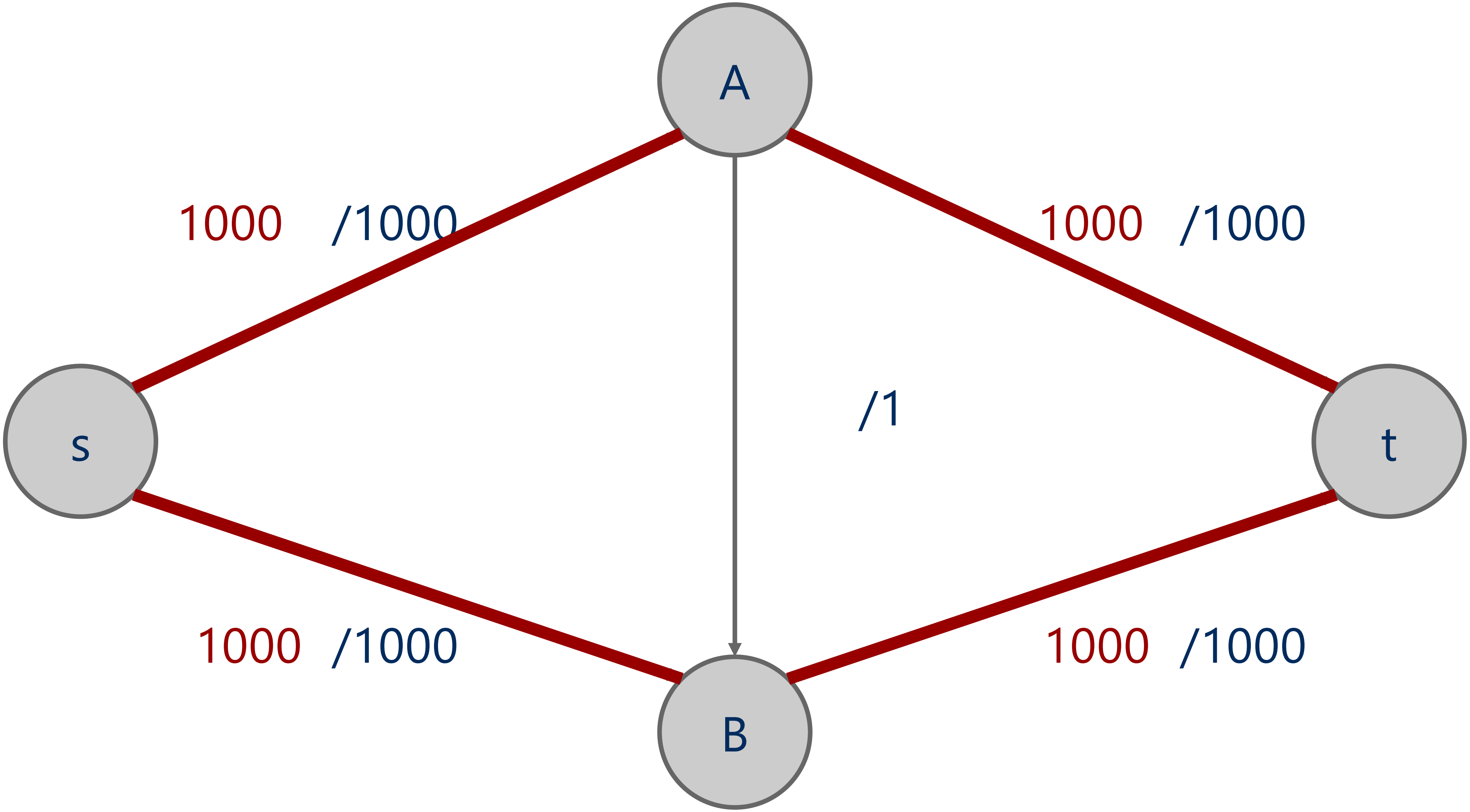
Max. Flow Value

time for finding an augmenting Path

Edmonds Karp

- How the augmenting path is chosen affects the performance of the search for max flow
- Edmonds and Karp proposed a shortest path heuristic for Ford Fulkerson
    - Use BFS to find augmenting paths

Another example

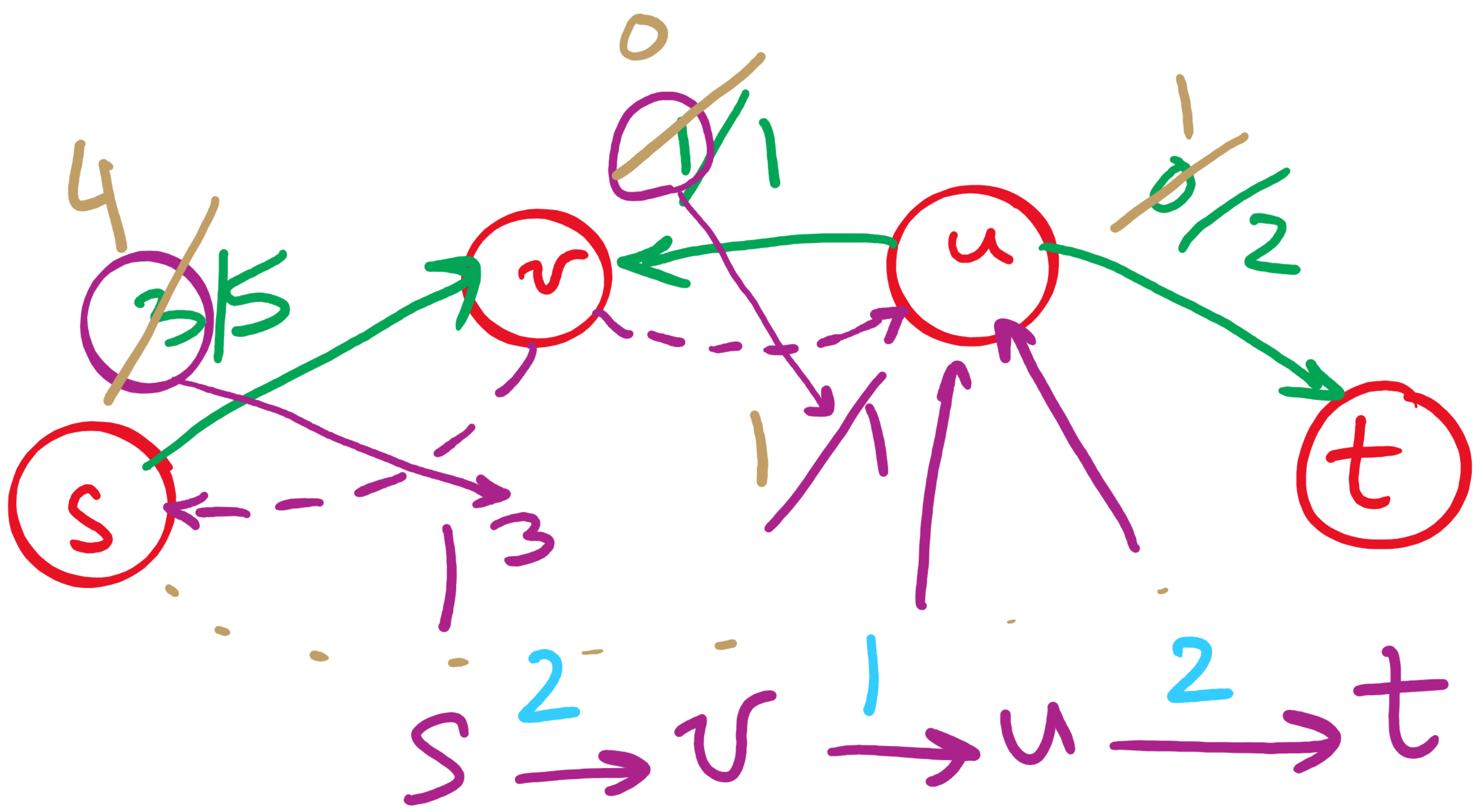


But our flow graph is weighted...

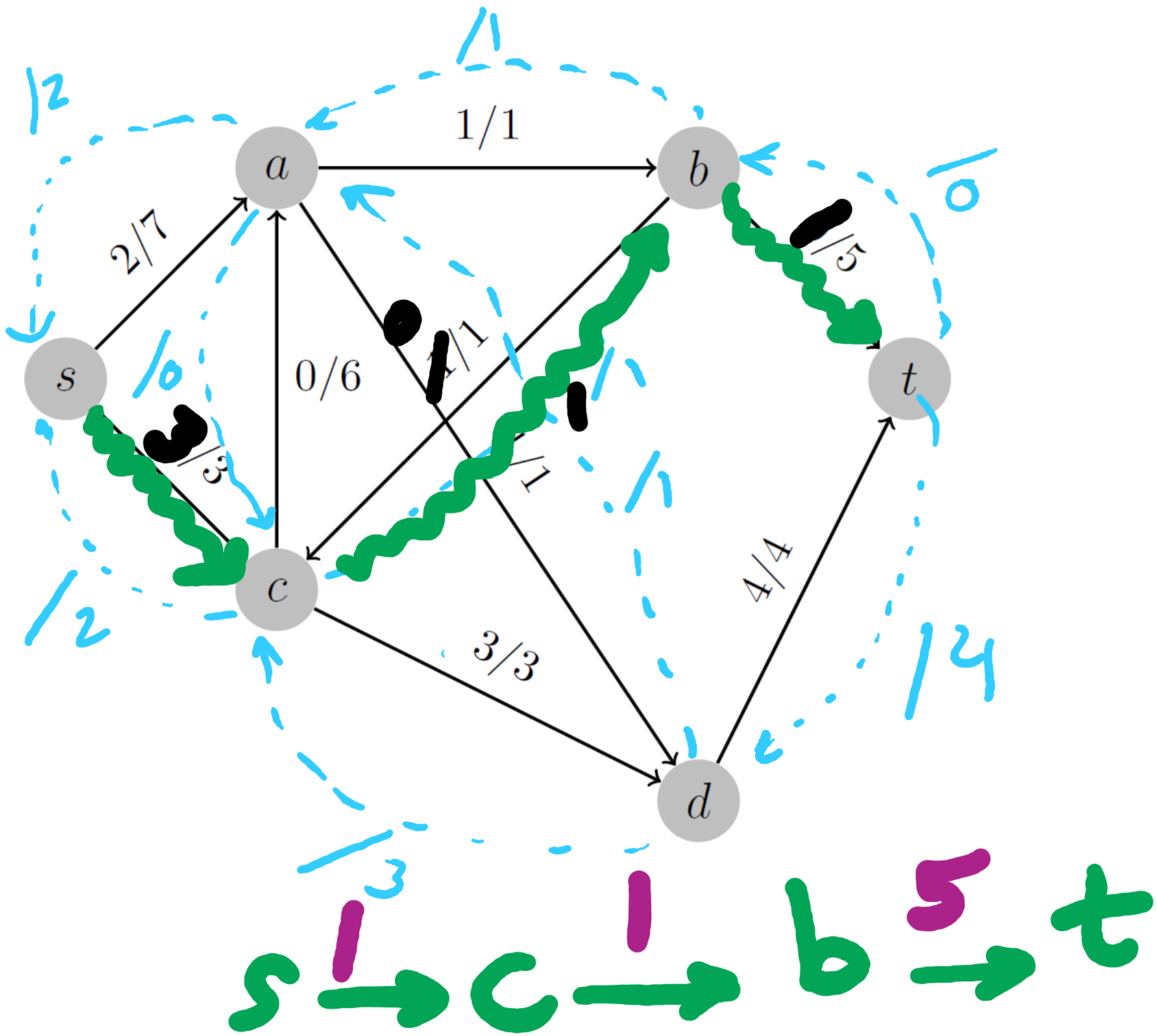
- Edmonds-Karp only uses BFS
  - Used to find spanning trees and shortest paths for *unweighted* graphs
  - Why do we not use some measure of priority to find augmenting paths?

Backwards edges

- Adding flow to a backwards edge means rerouting flow from the corresponding forward edge



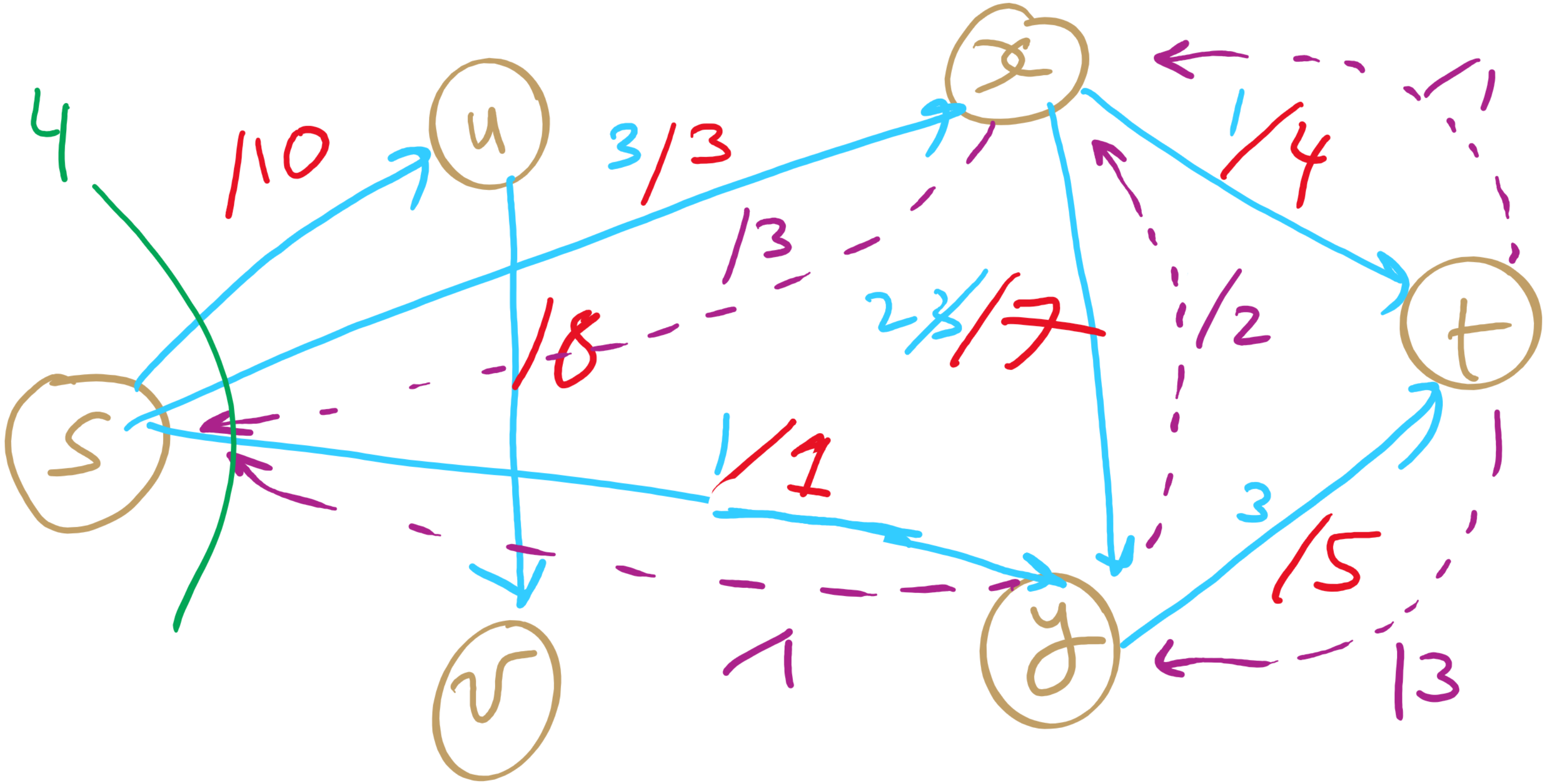
2nd Tophat Question



But our flow graph is weighted...

- Edmonds-Karp only uses BFS
  - Used to find spanning trees and shortest paths for *unweighted* graphs
  - Why do we not use some measure of priority to find augmenting paths?

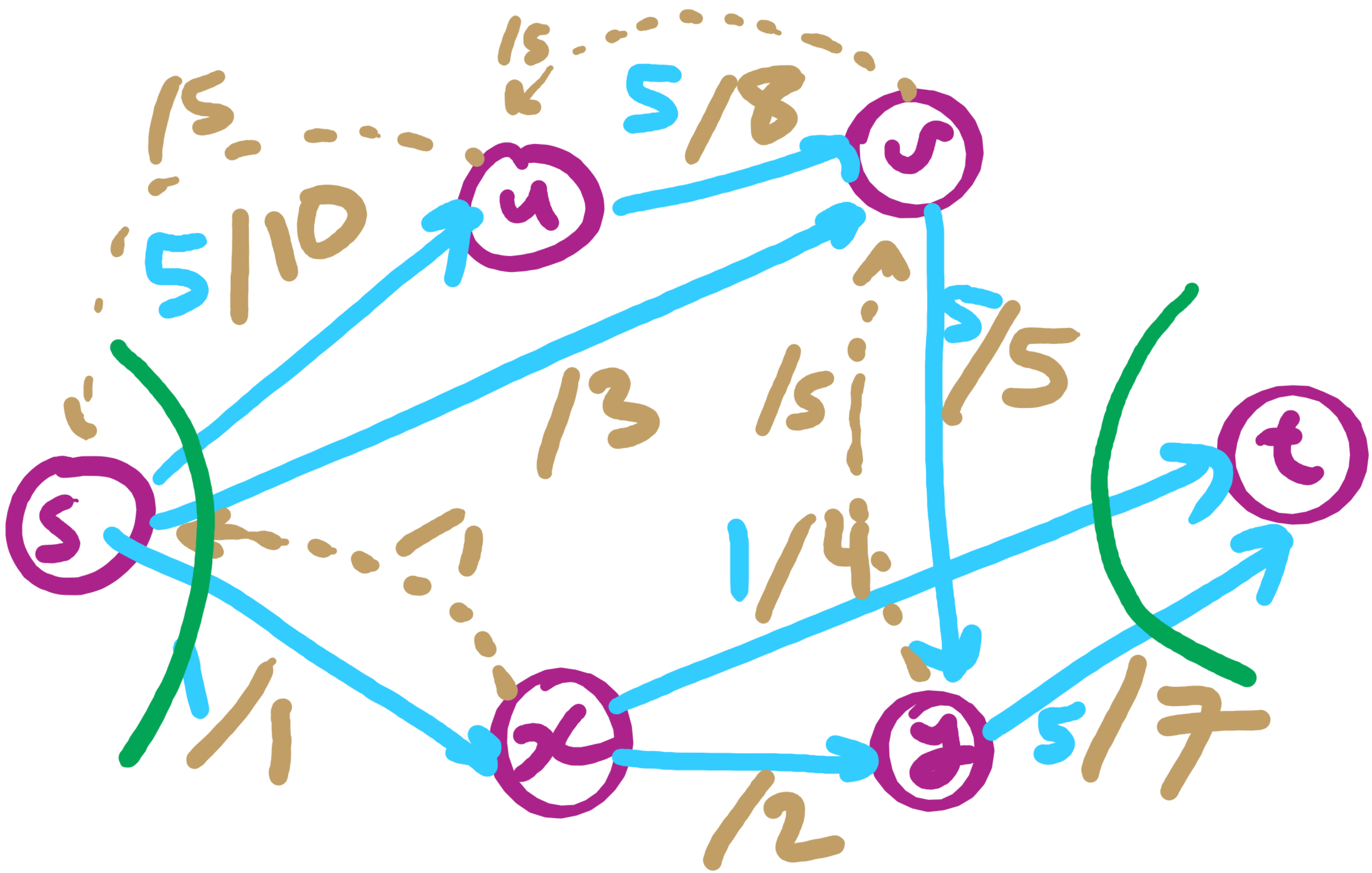
PFS Example 1



s → x → y → t

s → y → x → t

PFS Example 2



Flow edge implementation

- For each edge, we need to store:
    - Start point, the from vertex
    - End point, the to vertex
    - Capacity
    - Flow
    - Residual capacities
        - For forwards and backwards edges

FlowEdge.java

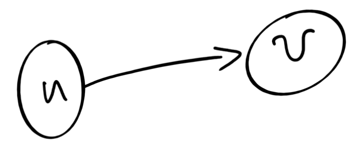
```
public class FlowEdge {
   private final int v;              // from
   private final int w;              // to
   private final double capacity;    // capacity
   private double flow;              // flow
     …
     public double residualCapacityTo(int vertex) {
      if       (vertex == v) return flow;
      else if (vertex == w) return capacity - flow;
      else throw new
       IllegalArgumentException("Illegal endpoint");
   }
   …
}
```

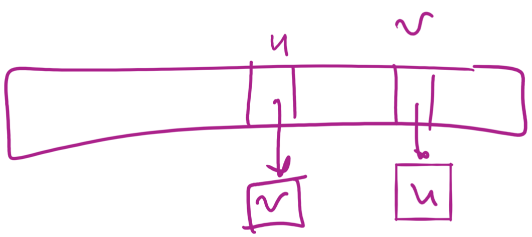
BFS search for an augmenting path (pseudocode)

```
edgeTo = [|V|]
marked = [|V|]
Queue q
q.enqueue(s)
marked[s] = true
while !q.isEmpty():
    v = q.dequeue()
    for each (v, w) in AdjList[v]:
        if residualCapacity(v, w) > 0:
            if !marked[w]:
                edgeTo[w] = v;
                marked[w] = true;
                q.enqueue(w);
```

Each FlowEdge object is stored in the adjacency list twice:

Once for its forward edge
Once for its backward edge





Value of maxflow

- Add up the flow increments in each iteration of Ford-Fulkerson
- Add up the edge **flows** out of source
- Add up the edge **flows** of the out of source

Follow-up Problem

- So, now we found the bottleneck *value*, but which edges define the found bottleneck?
    - *Why would you want to know those bottleneck edges?*

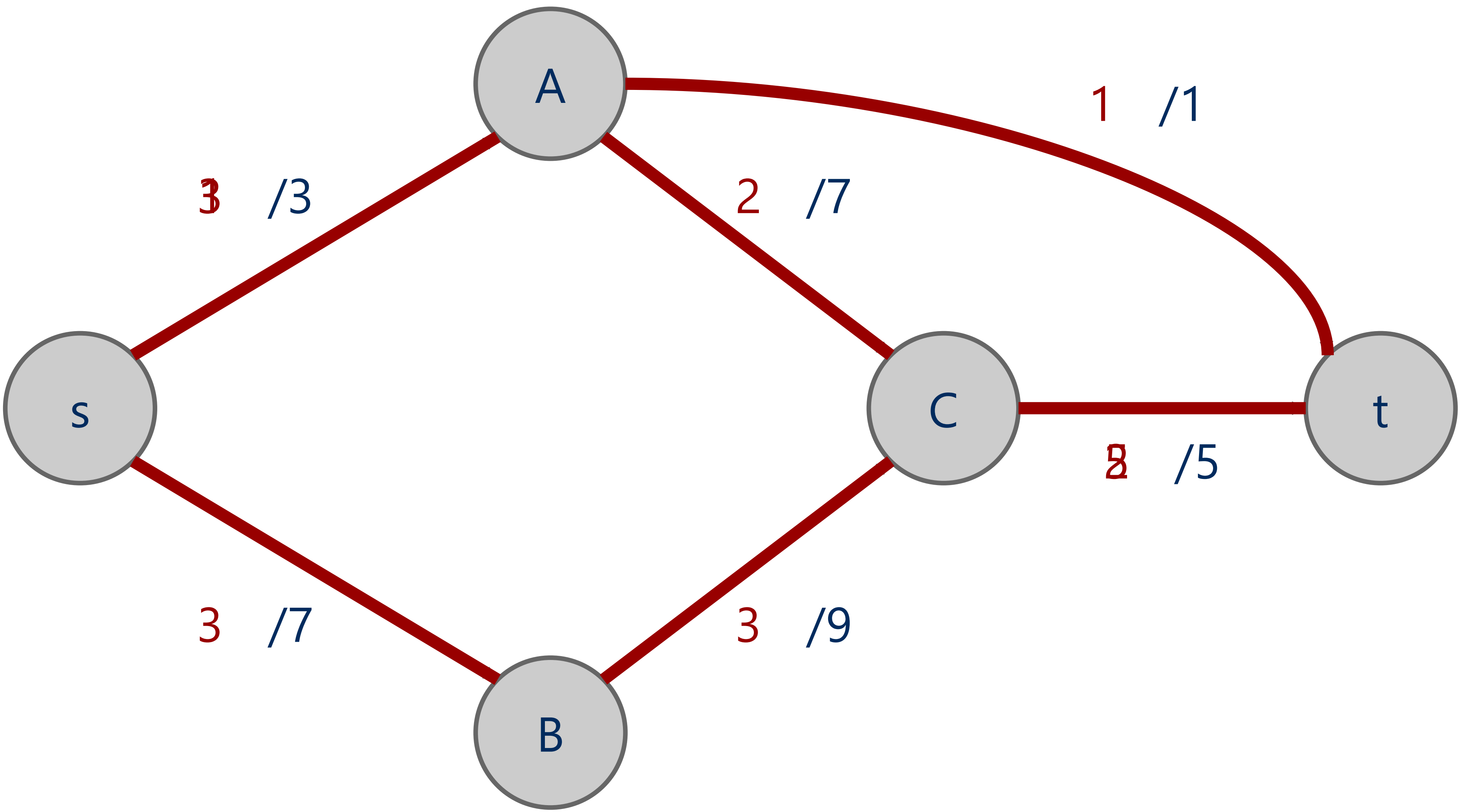
Let's separate the graph

- An st-cut on G is a set of edges in G that, if removed, will partition the vertices of G into two disjoint sets
  - One contains s
  - One contains t
- May be many st-cuts for a given graph
- Let's focus on finding the minimum st-cut
  - The st-cut with the smallest capacity
  - May not be unique

How do we find the min st-cut?

- We could examine residual graphs
  - Specifically, try and allocate flow in the graph until we get to a residual graph with no existing augmenting paths
    - A set of saturated edges will make a minimum st-cut

Min cut example



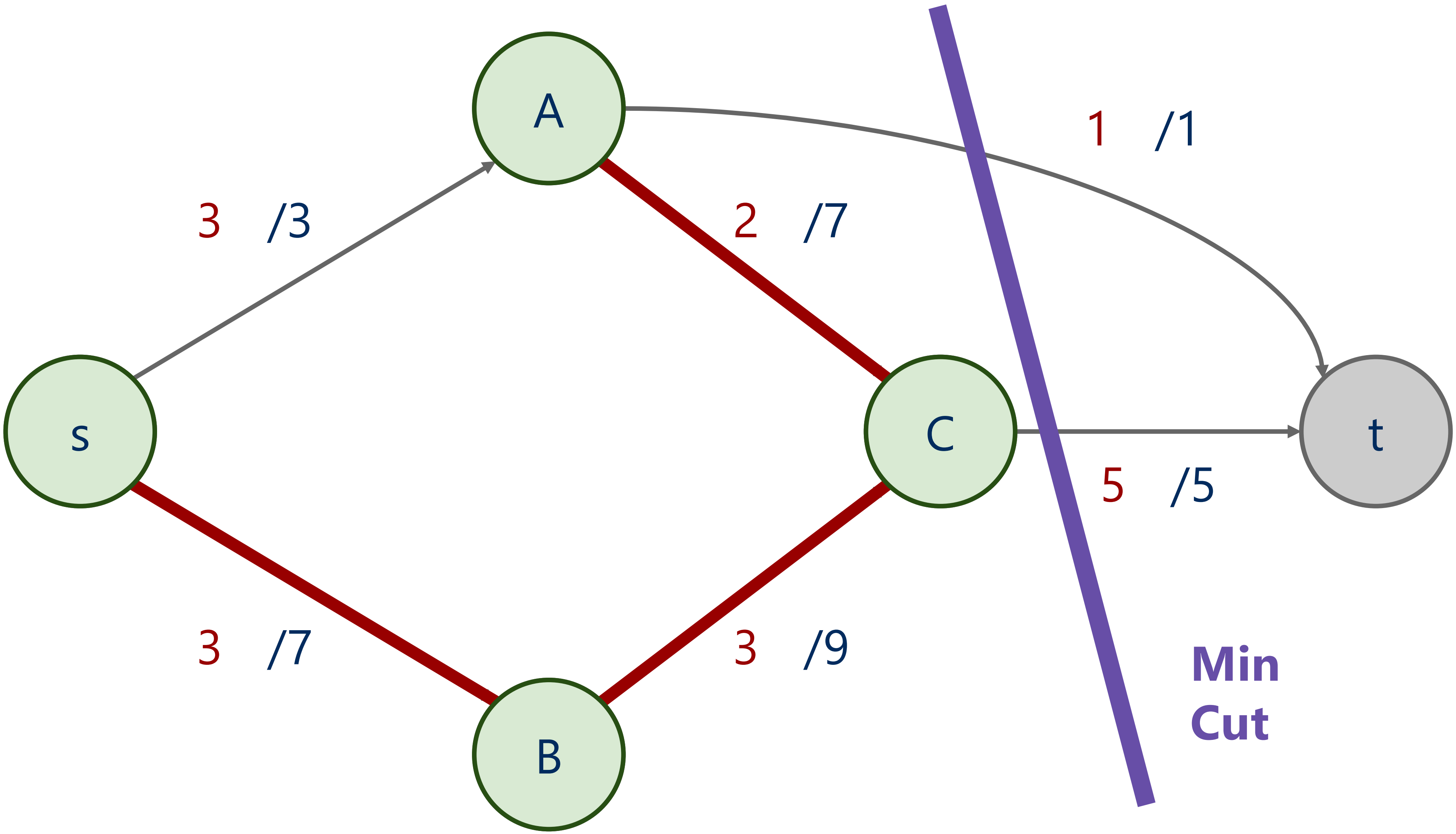
Max flow == min cut

- A special case of duality
  - I.e., you can look at an optimization problem from two angles
    - In this case to find the maximum flow or minimum cut
  - In general, dual problems do not have to have equal solutions
    - The differences in solutions to the two ways of looking at the problem is referred to as the *duality gap*
      - If the duality gap = 0, strong duality holds
        - Max flow/min cut uphold strong duality
      - If the duality gap > 0, weak duality holds

Determining a minimum st-cut

- First, run Ford Fulkerson to produce a residual graph with no further augmenting paths
- The last attempt to find an augmenting path will visit every vertex reachable from s
    - Edges with only one endpoint in this set comprise a minimum st-cut

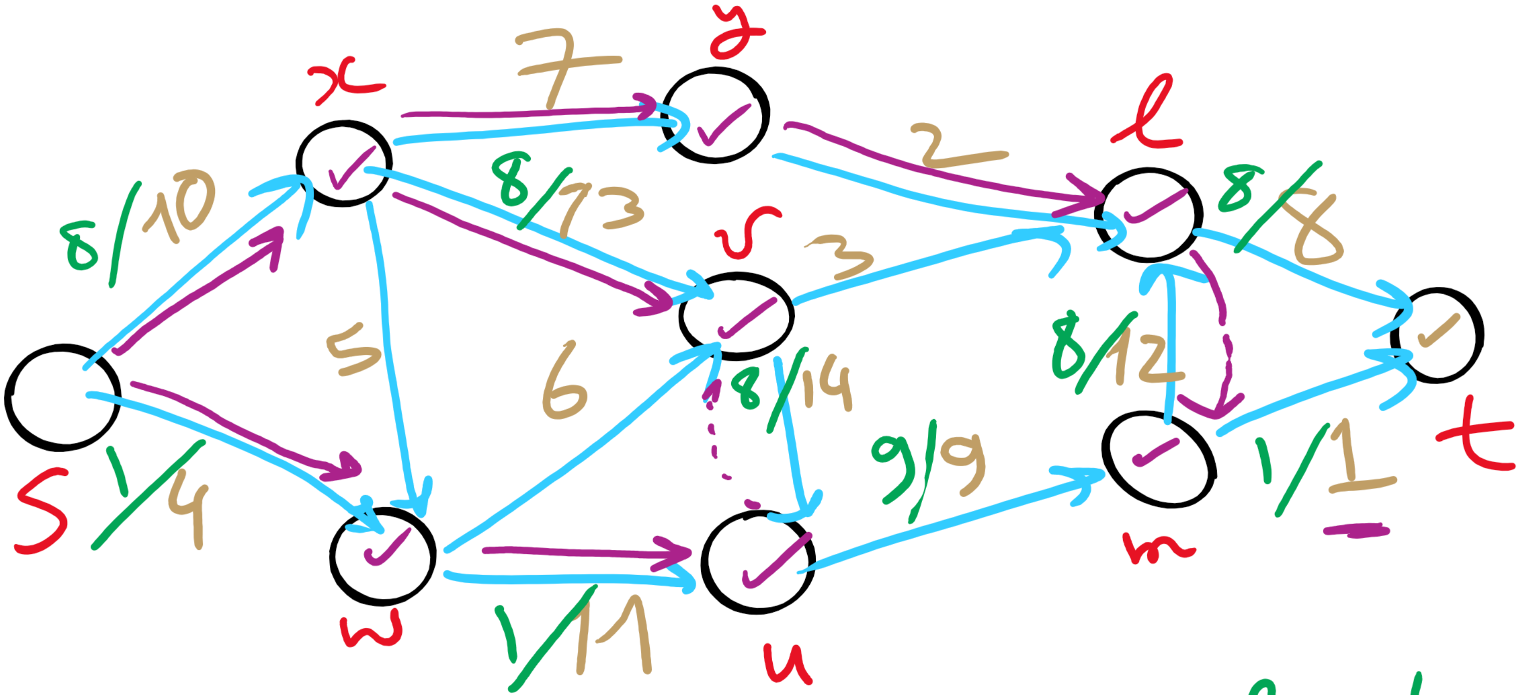
Determining the min cut



Max flow / min cut on unweighted graphs

- Is it possible?
- How would we measure the Max flow / min cut?
- What would an algorithm to solve this problem look like?

Min st-cut Example 1

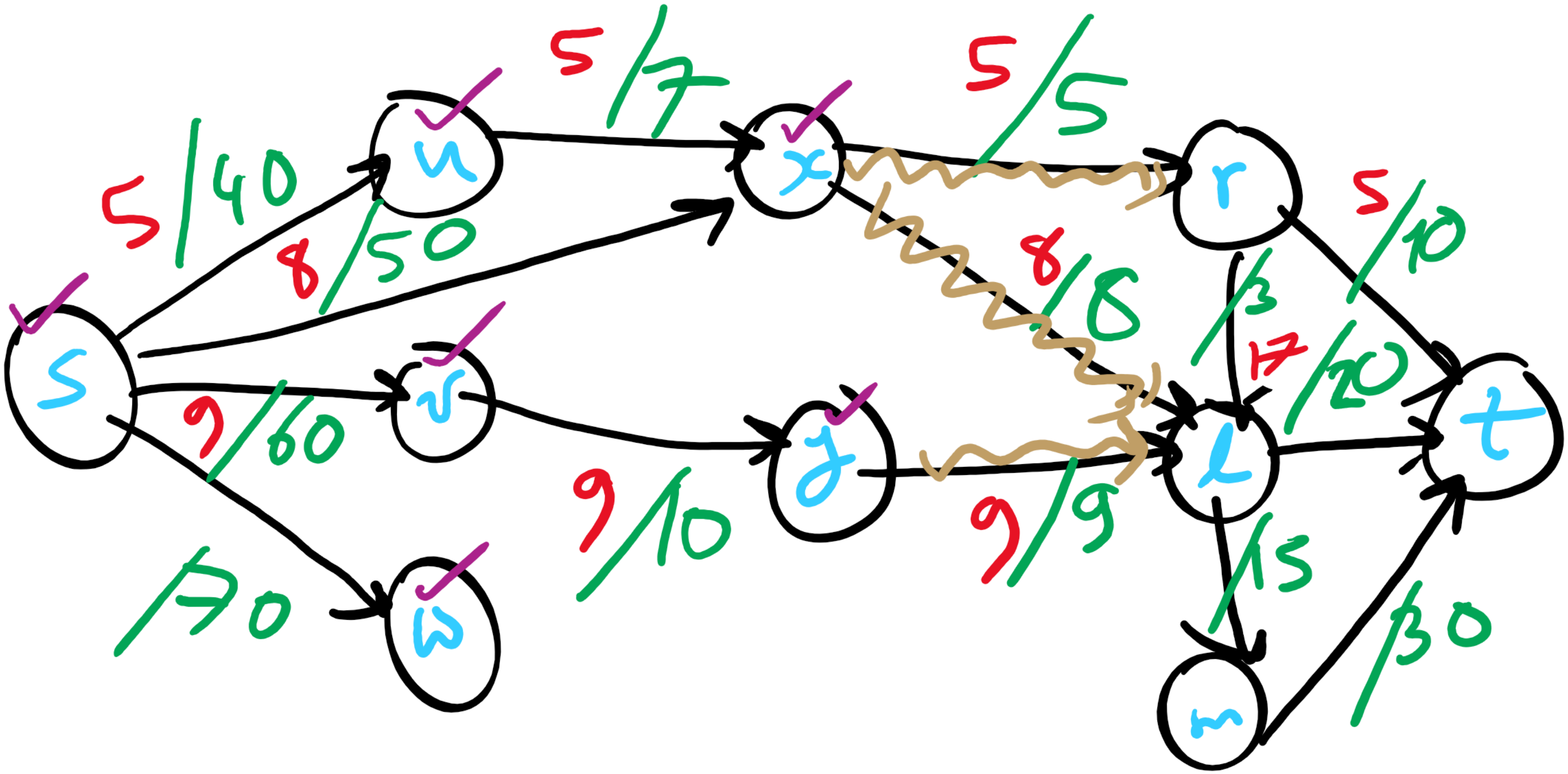


s → x → v → u → m → l → t

s → w → u → m → t

l → t
m → t } a min-cut

Min st-cut Example 2



$s \to u \to x \to r \to t$

$s \to v \to y \to l \to t$

$s \to x \to l \to t$

Unweighted network flow